

JDBC代码重构

1、JDBC操作模板：jdbc Template

- DML操作模板
 - 第一步：链接数据源
- utils包下的JDBCTemplate类

```
public class JDBCUtils {
    private static String driverClassName;
    private static String url;
    private static String username ;
    private static String password ;
    private static int maxActive;
    private static int initSize;
    private static DruidAbstractDataSource druid= null;
    static {
        //读取文件，得到一个输入流
        //首先通过当前类获取自己的class对象，在获取类加载器，得到输入流
        InputStream inputStream =
JDBCUtils.class.getClassLoader().getResourceAsStream("druid.properties");
        //创建Properties对象(属性对象)
        Properties properties = new Properties();
        try {
            //将属性流加载到属性对象中
            properties.load(inputStream);
            //获取其中内容
            druid = new DruidDataSource();//创建一个数据源
            //设置连接属性
            driverClassName = properties.getProperty("driverClassName");
            url = properties.getProperty("url");
            username = properties.getProperty("username");
            password = properties.getProperty("password");
            initSize = Integer.valueOf(properties.getProperty("initialSize"));
            maxActive = Integer.valueOf(properties.getProperty("maxActive"));
            druid.setDriverClassName(driverClassName);
            druid.setUrl(url);
            druid.setUsername(username);
            druid.setPassword(password);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static Connection getConnection(){
        Connection connection = null;
        try {
            connection = druid.getConnection();
        } catch (SQLException e) {
```

```

        e.printStackTrace();
    }
    return connection;
}

public static void releaseSource(ResultSet resultSet, Statement statement,
Connection connection){
    try {
        if (resultSet != null){
            resultSet.close();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }finally {
        try {
            if (statement != null){
                statement.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }finally {
            try {
                if (connection != null){
                    connection.close();
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
}
}
}
}

```

- 第二部：进行测试 test包 下的druidTest类

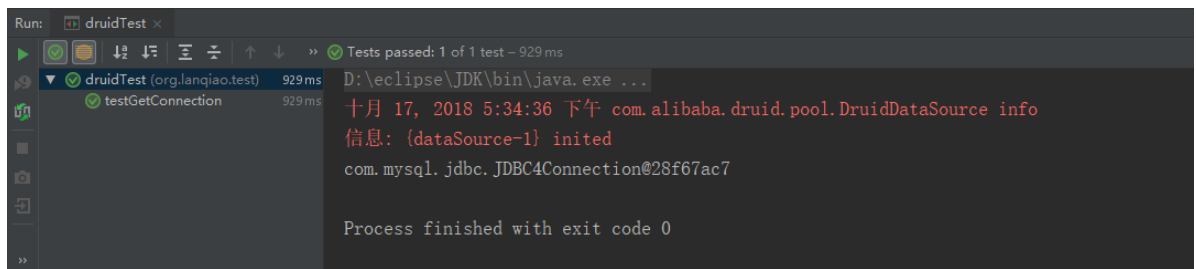
```

public class druidTest {

    @Test
    public void testGetConnection(){
        Connection connection = JDBCUtils.getConnection();
        System.out.println(connection);
    }
}

```

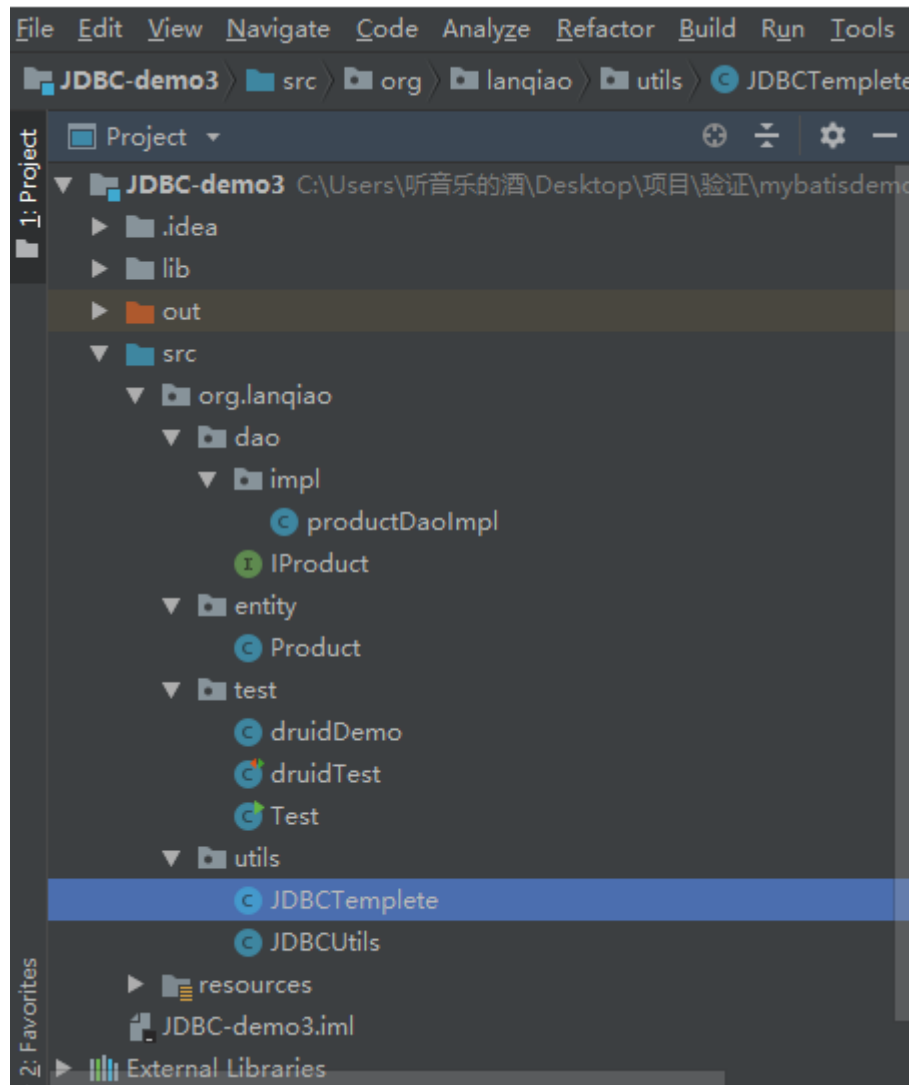
链接成功：如图



- 第三部：进行增删改操作

- 项目的目录

如图所示：



- 各个包下的具体实现

- utils包 下的JDBCTemplate类

```
//增删改操作
//参数: insert,delete,update
public static void execUpdate(String sql,Object ... args){
    Connection connection = JDBCUtils.getConnection();
    PreparedStatement preparedStatement = null;
    try {
        connection.setAutoCommit(false);//手动关闭
```

```

        preparedStatement = connection.prepareStatement(sql); //此处传哪个sql进来? 就执行哪个sql
        //设置参数
        //对参数进行for循环
        for(int i = 0; i < args.length; i++){
            preparedStatement.setObject(i+1, args[i]); //参数从哪开始
        }
        preparedStatement.executeUpdate(); //进行更新操作, 更新操作都需要事务

        connection.commit(); //手动提交
    } catch (SQLException e) {
        try {
            connection.rollback(); //若出现异常则进行回滚操作
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
        e.printStackTrace();
    } finally {
        //释放资源

        JDBCUtils.releaseSource(null, preparedStatement, connection);
    }
}

```

■ test包下的Test类

```

public class Test {
    public static void main(String[] args) throws Exception{
        IProduct iProductp = new productDaoImpl();
        /*//根据ID查询
        Product product = iProductp.getProductById(1);
        System.out.println(product);
        System.out.println("-----");
        //根据产品名称查询
        List<Product> productList = iProductp.getProductByName("M");
        for (Product product1:productList){
            System.out.println(product1);
        }
        System.out.println("-----");
        //查询所有
        List<Product> productAll = iProductp.getAll();
        for (Product productList2:productAll){
            System.out.println(productList2);
        }
        System.out.println("-----");
        //更新
        Product product3 = iProductp.getProductById(1);
        product3.setSalePrice(666);
        iProductp.updateProductById(product3);*/
        //删除
    }
}

```

```

        iProductp.deleteProductById(10);
    }
}

```

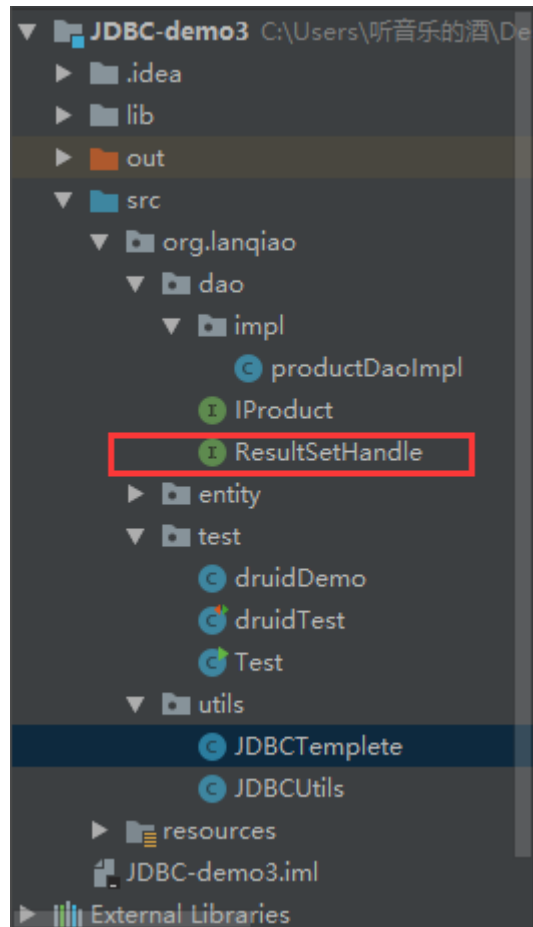
- 当当
 - 当当
- DQL操作模板
 - 结果集处理
 - 具体实现代码
 - utils包下的JDBCTemplate类

```

//查询操作
public static <T>T execSelect(String sql, ResultSetHandle<T> rsh,
Object ... args){
    Connection connection = JDBCUtils.getConnection();
    PreparedStatement preparedStatement = null;
    ResultSet resultSet = null;
    try {
        preparedStatement = connection.prepareStatement(sql);
        //传递参数(给它一个参数)
        //对参数进行循环(参数设置,Object ... args[])
        for(int i = 0;i < args.length;i++){
            preparedStatement.setObject(i + 1,args[i]);
        }
        //返回一个结果集
        resultSet = preparedStatement.executeQuery();
        //进行结构集处理
        //新建一个处理结果集的接口
        //添加一个结果集处理器
        //结果集的处理
        T t =rsh.handle(resultSet);//得到一个泛型对象
        return t;
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}

```

- 在Dao层下添加一个接口
 - 项目结构目录



- 代码

```
public interface ResultSetHandle<T> {  
    T handle(ResultSet resultSet) throws SQLException; //结果集处理后，返回一个T对象  
}
```

- 当当

- test包下的Test类

```
//基于DQL操作模板的查询  
Product product = iProductp.getProductById(1);  
System.out.println(product);
```

- 方法泛型参数设计

- 返回结果

- 对象
- 集合
- 使用统计函数返回一个单个数字
- 提供了一个泛型方法，根据不同结果集提供不同实现类
- 如图：

实现了ResultHandle<T>的处理结果集类

1：处理list结果集

```
public class BeanListHandle<T> implements ResultHandle<List<T>> {  
    private Class<T> clazz;  
    public BeanListHandle(Class<T> clazz) {  
        this.clazz=clazz;  
    }  
    @Override  
    public List<T> handle(ResultSet rs) {  
        @Override  
        public List<T> handle(ResultSet rs) {  
            List<T> list=new ArrayList<T>();  
            try {  
                while(rs.next()){  
                    T t=clazz.newInstance();  
                    BeanInfo beanInfo = Introspector.getBeanInfo(clazz, Object.class);  
                    PropertyDescriptor[] pds = beanInfo.getPropertyDescriptors();  
                    for (PropertyDescriptor pd : pds) {  
                        for (PropertyDescriptor pd : pds) {  
                            String name = pd.getName();  
                            Method method = pd.getWriteMethod();  
                            method.invoke(t,rs.getObject(name));  
                        }  
                    }  
                    list.add(t);  
                }  
            }  
        }  
    }  
}
```

```

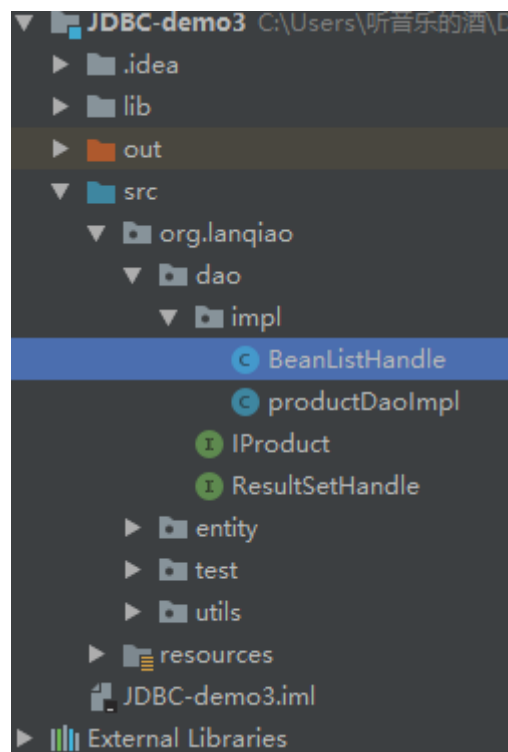
    } catch (Exception e) {
        e.printStackTrace();
    }

    return list;
}
}

```

2：处理查询数量的结果集

- 新建一个类：BeanListHandle（dao层下的Impl包中）
 - 如图：项目结构图



- 代码

```

public class BeanListHandle<T> implements ResultSetHandle<List<T>>{
    //定义成员属性
    private Class<T> clazz;
    //构造方法
    public BeanListHandle(Class<T> clazz){
        this.clazz = clazz;
    }
}

```



```

    }

    //实现接口的方法
    @Override
    public List<T> handle(ResultSet resultSet) throws SQLException,
        IllegalAccessException, InstantiationException,
        IntrospectionException, InvocationTargetException {
        //进行结果集的处理
        //1、创建一个List
        List<T> list = new ArrayList<>();
        while (resultSet.next()){
            //创建一个对象
            T t = clazz.newInstance();
            //获取一个包含属性的对象
            BeanInfo bean =
                Introspector.getBeanInfo(clazz, Object.class);
            //得到一个数组
            PropertyDescriptor[] pds = bean.getPropertyDescriptors();
            //对数组进行for循环
            for(int i = 0; i < pds.length; i++){
                //获取属性名称
                String name = pds[i].getName();
                //获取所有写的方法
                Method method = pds[i].getWriteMethod();
                //执行
                method.invoke(t, resultSet.getObject(name));
            }
            list.add(t);
        }
        return list;
    }
}

```

- 修改productDaoImpl类中getProductById(int id)方法的代码

```

@Override//根据ID查询
public Product getProductById(int id) {
    String sql = "select * from product where id = ?";
    Product product = new Product();
    List<Product> list = JDBCTemplate.execSelect(sql, new
        BeanListHandle<>(Product.class) , id);
    return list.get(0);
}

```

- 当当

- 当当

- 当当

2、函数

3、当当